clock scan

clock scan scans a string and extracts a time.

See also

Documentation

official reference

Synopsis

clock scan *inputString* ?*-option value...*?

Description

Parses the time described in *inputString* and returns the number of seconds (since the start of the Unix epoch) that that time corresponds to. Supported *-option*s are:

-base *time*

-format *format*

-gmt *boolean*

-locale *localeName*

-timezone *zoneName*

Incomplete times

[clock scan] does what it can to make sense of incomplete times, but the results can be surprising. If possible, make sure at least the year, month, and day are part of the string to scan. If the results of scanning an incomplete or impossible date like March 0 or April 34 are surprising, carefully re-read the seven rules that are applied, in order of preference, to make sense of a time.

TCV 2013-02-11: providing only a year (e.g. %Y) in the format string will *not* produce the time for the first day of that year. Similarly, providing only the year and month (e.g. %m/%Y) in the format string will *not* produce the time for the first day of that month. In both of these cases, you will instead get the time corresponding to midnight on the current day. Currently in clock scan, to avoid this behaviour, day level accuracy is required; this can be done by specifying:

- year and day-of-year
- week and day-of-week

- month and day-of-month

So in the examples above, just adding to the input string 01 for each of the missing fields will give you the time you're interested in (first day of the year or month, respectively).

Examples:

```
# Things you might think would behave differently.

% clock scan 2000 -format %Y
1360558800
% clock scan 10/1900 -format %m/%Y
1360558800

# They're actually midnight today!

% clock format [clock seconds] -format %Y-%m-%d
2013-02-11
% clock format 1360558800 -format %Y-%m-%d %H:%M:%S
2013-02-11 00:00:00

# A simple solution. That's what I really meant!

% clock scan "2000-01-01" -format "%Y-%m-%d"
946702800
% clock scan "1900-10-01" -format "%Y-%m-%d"
-2185387200
```

Removing /etc/localtime

JMN 2007-11-14:

On FreeBSD - you can configure the timezone to UTC by removing /etc/localtime. I don't know if this is actually a legitimate way to set it to UTC, but it was the only way I knew of until now, and it does seem to cause a problem with Tcl's clock scan.

```
% clock scan "2007-11-01" -format "%Y-%m-%d"
time value too large/small to represent
% clock scan "2007-11-01" -format "%Y-%m-%d"
1193875200
```

The call always fails the first time in each process, and seems to work for all calls thereafter. (including in other interps)

If you configure the timezone to UTC by instead making /etc/localtime a link to /usr/share/zoneinfo/Etc/UTC the problem doesn't occur. I have a number of systems configured with no /etc/localtime - but I guess I'll change that now in light of this.

This issue also shows up in the FAQ on http://amsn-project.net/ .

Is a missing /etc/localtime something that clock scan should be able to handle - or is it would it be considered an OS configuration issue?

DKF: Ultimately, I'd consider this to be OS misconfiguration as it feels like stating that the system has *no* local timezone instead of that it is UTC.

RCS timestamp problem

AMG: [clock scan] is behaving badly for me when processing RCS timestamps:

```
% clock format [clock scan "2000/11/01 06:37:54"]
Tue Aug 11 06:37:54 CST 2167
```

Why so far off? This date should be interpreted as 2000-Nov-01. Let's try that again:

```
% clock format [clock scan "2000/11/01 06:37:54" -format "%Y/%m/%d %T"]
Wed Nov 01 06:37:54 CST 2000
```

Incremental clock scan

dbohdan 2014-07-14: I wrote the following proc to deal with the surprising behavior of clock scan when given the date format of %Y or %Y-%m and the deprecation of free-format clock scan.

```
# Try several formats for clock scan.
proc incremental-clock-scan {date {debug 0}} {
    set date [regsub -all {[ :.T/]+} $date {-}]

    set result {}
    foreach {format padding} {
        {%Y} {-01-01-00-00-00}
        {%Y-%m} {-01-00-00-00}
        {%Y-%m-%d} {-00-00-00}
        {%Y-%m-%d-%H-%M} {-00}
        {%Y-%m-%d-%H-%M-%S} {}
    } {
        if {$debug} {
            puts "$format $date"
        }
        if {![catch {
                set scan [clock scan $date -format $format]
            }]} {
            # Work around unexpected treatment %Y and %Y-%m dates, see
            # https://wiki.tcl-lang.org/2525.
            set result [
                clock scan [
                    join [
                        list $date $padding
                    ] ""
                ] -format {%Y-%m-%d-%H-%M-%S}
            ]
            if {$debug} {
                puts "match"
                puts [clock format $scan]
            }
        }
    }
    return $result
}
```

It assumes the unspecified month or day to be 01 and the unspecified time to be midnight and treats the delimiters :.T/ as equivalent, so

- incremental-clock-scan {2014}
- incremental-clock-scan {2014-01}
- incremental-clock-scan {2014-01-01T00:00:00}
- incremental-clock-scan {2014/01/01 00:00:00}

all return the same value as clock scan {2014-01-01-00-00-00} -format {%Y-%m-%d-%H-%M-%S}.

Date range

RS 2017-05-04 - The **limits of timestamps** since Tcl 8.5 are less constrained than they used to be in earlier versions, where 2037 would be the last possible year...

Now we can go farther into the future (tested with a clock scan/clock format combo - OK if input day matches output day):

```
% clock format [clock scan 2525-09-13] ;# Zager & Evans, iirc
Thu Sep 13 00:00:00 CET 2525
```

The past works well until Sep. 14, 1752:

```
% clock format [clock scan 1752-09-14]
Thu Sep 14 00:00:00 CET 1752
```

but fails before that day:

```
% clock format [clock scan 1752-09-13]
Sun Sep 24 00:00:00 CET 1752
```

It just so happens that the Gregorian calendar was introduced in Scotland, England, and its colonies in 1752 - possibly just on September 14... ;^)

Anchorage

AMG: [clock scan] does not support the Anchorage timezones [L1], even though [clock format] does.

```
% clock format [clock seconds] -timezone :America/Anchorage
Mon May 21 11:53:41 AKDT 2018
% clock scan "Mon May 21 11:53:41 AKDT 2018"
unable to convert date-time string "Mon May 21 11:53:55 AKDT 2018": syntax error
(characters 19-23)
% clock scan "Mon May 21 11:53:55 AKDT 2018" -format "%a %b %d %H:%M:%S %z %Y"
time zone "akdt" not found
% dict set ::tcl::clock::LegacyTimeZone akdt -0800
% clock scan "Mon May 21 11:53:55 AKDT 2018" -format "%a %b %d %H:%M:%S %z %Y"
1526932435
```

Daylight Saving Time

BrucePreston: [clock scan] does not handle ambiguities arising from the end of daylight saving time consistently.

The manual page states under SCANNING TIMES that:

If a format string lacks a %z or %Z format group, it is possible for the time to be ambiguous because it appears twice in the same day, once without and once with Daylight Saving Time. If this situation occurs, the first occurrence of the time is chosen.

This is true sometimes:

```
% clock format 1541307600 -timezone :America/New_York
Sun Nov 04 01:00:00 EDT 2018
% clock format 1541311200 -timezone :America/New_York
Sun Nov 04 01:00:00 EST 2018
% clock scan "2018-11-04 01:00:00" -format "%Y-%m-%d %T" -timezone
:America/New_York
1541307600
```

..but not always:

```
% clock format 1540684800 -timezone :Europe/London
Sun Oct 28 01:00:00 BST 2018
% clock format 1540688400 -timezone :Europe/London
Sun Oct 28 01:00:00 GMT 2018
% clock scan "2018-10-28 01:00:00" -format "%Y-%m-%d %T" -timezone :Europe/London
1540688400
```

Curious syntax

AMG: The legacy [clock scan] parser allows some very curious syntax.

```
clock scan "next last thiss now tomorrow ago"
```

works the same as:

```
clock scan yesterday
```

I have to write thiss because the parser is buggy and strips off one of the s characters at the end because s means plural [L2] [L3].

Start of today

AMG: There does not appear to be a direct way to get the start of the current day. The start of the previous and next days are easy; just say yesterday or tomorrow. But today means now. Here's one possible solution:

```
clock add [clock scan yesterday] 1 day
clock format [clock add [clock scan yesterday] 1 day]
```

When not using the default timezone, for example :UTC, don't forget to tell both [clock scan] and clock format the timezone.

```
clock add [clock scan yesterday -timezone :UTC] 1 day
clock format [clock add [clock scan yesterday] 1 day] -timezone :UTC
```

I know it's also possible to format the current date then scan it again, but I imagine this is less efficient than the above.

```
clock scan [clock format [clock seconds] -format %D]
```

However, the time command shows that I'm completely wrong about this. The clock add method takes 2.8 times as long as the clock format method.

Again, don't forget about timezones.

```
clock scan [clock format [clock seconds] -format %D -timezone :UTC] -timezone :UTC
```

Remainder of today

AMG: To find how many seconds are left in the current day, try:

```
expr {[clock scan tomorrow] - [clock seconds]}
```

If dealing with a different timezone, use the -timezone switch to [clock scan].

See clock seconds for an alternate method that only samples the system time once rather than twice and thus will be robust even when those two samples fall on opposite sides of midnight.

Antiquities

AMG: [clock scan] doesn't like years before 0100 AD. Year 0099 and before are treated as if the 00 were 19.

```
% clock format -59011632000 -timezone :UTC
Wed Jan 01 00:00:00 UTC 0100
% clock scan "Wed Jan 01 00:00:00 UTC 0100"
-59011632000
% clock format -59011632001 -timezone :UTC
Tue Dec 31 23:59:59 UTC 0099
% clock scan "Tue Dec 31 23:59:59 UTC 0099"
946684799
% clock format 946684799 -timezone :UTC
Fri Dec 31 23:59:59 UTC 1999
```